Command Line Interface, Stochastic SVD*

Contents

*Dmitriy Lyubimov, dlyubimov at apache dot org

1 Background information

1.1 Stochastic SVD (SSVD)

Stochasitc SVD method in Mahout produces reduced rank Singular Value Decomposition output in its strict mathematical definition:

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

i. e. it creates outputs for matrices $\mathbf{U}$, $\mathbf{V}$ and $\mathbf{\Sigma}$, each of which may be requested individually. The desired rank of decomposition, henceforth denoted as $k \in \mathbb{N}_1$, is a parameter of the algorithm. The singular values inside diagonal matrix $\mathbf{\Sigma}$ satisfy $\sigma_{i+1} \leq \sigma_i \; \forall i \in [1, k-1]$, i.e. sorted from biggest to smallest. Cases of rank deficiency rank $(\mathbf{A}) < k$ are handled by producing 0s in singular value positions once deficiency takes place.

Single space for comparing row-items and column-items. On top of it, there's an option to present decomposition output in a form of

$$\mathbf{A} \approx \mathbf{U}_\sigma \mathbf{V}_\sigma^\top, \tag{1}$$

where one can request $\mathbf{U}_\sigma = \mathbf{U}\mathbf{\Sigma}^{0.5}$ instead of $\mathbf{U}$ (but not both), $\mathbf{V}_\sigma = \mathbf{V}\mathbf{\Sigma}^{0.5}$ instead of $\mathbf{V}$ (but not both). Here, notation $\mathbf{\Sigma}^{0.5}$ implies diagonal matrix containing square roots of the singular values:

$$\mathbf{\Sigma}^{0.5} = \begin{pmatrix} \sqrt{\sigma_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\sigma_k} \end{pmatrix}.$$

Original singular values $\mathbf{\Sigma}$ are still produced and saved regardless.

This option is a nod to a common need of comparing actors represented by both input rows and input columns in a common space. E.g. if LSI is performed such that rows are documents and columns are terms then it is possible to compare documents and terms (ether existing or fold in new ones) in one common space and perform similarity measurement between a document and a term, rather than computing just term2term or document2document similarities.

Folding in new observations. It is probably worth mentioning the operation of "folding in" new observations in context of this method, since it is often a basis for incremental methods.

If $\tilde{\mathbf{c}}_r$ ($\tilde{\mathbf{c}}_c$) is a new row (column) observation in addition to original input $\mathbf{A}$, then correspondent "new" row vectors of $\tilde{\mathbf{U}}$ $\left(\tilde{\mathbf{V}}\right)$ can be obtained as

$$\tilde{\mathbf{u}} = \mathbf{\Sigma}^{-1}\mathbf{V}^\top\tilde{\mathbf{c}}_r, \tag{2}$$
$$\tilde{\mathbf{v}} = \mathbf{\Sigma}^{-1}\mathbf{U}^\top\tilde{\mathbf{c}}_c. \tag{3}$$

Similarly, for the case (1) folding in new observations into rows of $\tilde{\mathbf{U}}_\sigma$ $\left(\tilde{\mathbf{V}}_\sigma\right)$ would look like

$$\tilde{\mathbf{u}}_\sigma = \mathbf{V}_\sigma^\top\tilde{\mathbf{c}}_r, \tag{4}$$
$$\tilde{\mathbf{v}}_\sigma = \mathbf{U}_\sigma^\top\tilde{\mathbf{c}}_c. \tag{5}$$

Thus, new rows can be added to matrices denoted as $\tilde{\mathbf{U}}$ $\left(\tilde{\mathbf{V}}\right)$ corresponding to new observations as new observations become available, i.e. incrementally. Given that new observations are usually moderately sparse vectors, it might be feasible to do fold-in in real time or almost real time, assuming proper fast row-wise indexing of $\mathbf{U}$ ($\mathbf{V}$) exists (e.g. using a batch request to an HBase table containing rows of $\mathbf{U}$ ($\mathbf{V}$)). However, since operation of folding in new observations doesn't change original decomposition and its spaces, such new observations cannot be considered 'training' examples. Typically, from time to time accumulated new observations can be added to original input $\mathbf{A}$ and the whole decomposition can be recomputed again.

Common applications for SVD include Latent Semantic Analysis (LSA), Principal Component Analysis (PCA), dimensionality reduction and others.

A note about stochasticity and result precision. The case that needs to be specifically warned about is the case where the data has variances in a *great deal* of principal orthogonal directions *with no statistically signficant differences* in between them, i.e., the input data has *no interesting spectrum decay*. A matrix where each entry is generated from a random distribution with 0 mean, would be an example of such input data. Figuring principal data variances for such randomly generated and at the same time "big" data are hard and may result in sufficiently problematic errors in the final solution compared to optimal solution (if we could obtain it at that scale). But the cases

dominated by a random noise with no statistically significant signal are hopefully not the data we actually are forced to face with in practice. But even quite small signal buried in a sea of "big data" noise could probably eventually be dug out given enough computational effort with spent, with this method (by cranking up the `-q` setting) but will likely also eventually reach a point of diminishing return. The said is equally true for any n-th singular vector where spectrum decay sufficiently flattens out.

Another sufficiently not very interesting case are "problems too small" (see also §5.1). Optimal solution will be produced with $k+p = \mathrm{rank}(\mathbf{A})$ though, and stochastic errors will start appearing with $k+p < \mathrm{rank}\,(\mathbf{A})$. However errors will hopefully be more acceptable if $k+p \ll \mathrm{rank}\,(\mathbf{A})$.

1.2 PCA options in SSVD

Some of interesting applications of SVD is dimensionality reduction and Principal Component Analysis. As of MAHOUT-817, SSVD method is equipped with options helping to produce both PCA and dimensionality reduction transformations.

PCA is also one of the methods to achieve *dimensionality reduction*.

Outline: data points are row vectors. We approach general PCA and dimensionality reduction problem with respect to input expressed in Mahout's distributed row matrix format. We also assume data points are row vectors in such matrix[1]. We denote such $m \times n$ input matrix as $\mathbf{A}^{(pca)}$:

$$\mathbf{A}^{(pca)} = \begin{pmatrix} \boldsymbol{a}_1^\top \\ \boldsymbol{a}_2^\top \\ \vdots \\ \boldsymbol{a}_m^\top \end{pmatrix}$$

Column mean is n-vector[2]

$$\begin{aligned} \boldsymbol{\xi} &= \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{pmatrix} \\ &= \frac{1}{m}\sum_i^m \boldsymbol{a}_i. \end{aligned}$$

We denote $m \times n$ mean matrix as

$$\boldsymbol{\Xi} = \begin{pmatrix} \boldsymbol{\xi}^\top \\ \boldsymbol{\xi}^\top \\ \vdots \\ \boldsymbol{\xi}^\top \end{pmatrix} \in \mathbb{R}^{m\times n}$$

Traditional approach under these settings starts with finding a column mean $\boldsymbol{\xi}$ and subtract it from all data points (row vectors) of the input

$$\mathbf{A} = \mathbf{A}^{(pca)} - \boldsymbol{\Xi}$$

and then proceeds with finding a reduced k-rank SVD:

$$\mathbf{A} \approx \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top. \tag{6}$$

At this point rows of $\mathbf{U}$ (or, more strictly, rows of product $\mathbf{U}\boldsymbol{\Sigma}$) correspond to original data points (rows of $\mathbf{A}^{(pca)}$) converted to approximate PCA space[3] and $\mathbf{V}$ represents approximate eigenvectors of the covariance matrix of our input (and we are done with PCA part at this point). Note that since our datapoints in this case are row vectors in the input (and not column vectors), covariance matrix is $\mathbf{C} = \frac{1}{m}\mathbf{A}^\top\mathbf{A}$. [4]

[1] Note that in wikipedia article input data points are considered to be column vectors, so our input is transpose w.r.t. to case outlined in the wikipedia PCA article.

[2] In many cases in literature one would find PCA mean vector denoted as $\boldsymbol{\mu}$.

[3] Exact PCA space requires $\mathbf{AV}$ product but can be assumed as $\mathbf{AV} \approx \mathbf{U}\boldsymbol{\Sigma}$ because of (6)

[4] Some courses (see Andrew Ng's ML online lectures) also mention a metric aka "percent of variance retained" $\sum_{i=1}^k \sigma_i / \sum_{i=1}^n \sigma_i \cdot 100\%$ that seems to suggest that it would be a metric to compare "goodness" of choice for k . We of course cannot know all of the singular values of the full SVD in context of SSVD, although we can estimate that

$$\frac{\sum_{i=1}^k \sigma_i}{\sum_{i=1}^n \sigma_i} \geq \frac{\sum_{i=1}^k \sigma_i}{(n-k)\,\sigma_k + \sum_{i=1}^k \sigma_i},$$

i.e. we can make a statement in a sense that "we have *at least* that much variance retained" instead. Whether this estimate is useful or not in practice, is not clear to me as I have no error estimate (and, more importantly, no error estimate *with* stochastic error baked in) between expression on left and right of the inequality at the moment. But at least it puts some metric threshold on how big k we may need and identify some cases where smaller values of k will suffice given particular input and minimum variance retained prerequisite thus reducing need for flops. Another possible way to estimate it is perhaps to try and fit existing singular values into asymptotically decaying curve in order to produce a better guess for $\sum_{i=k+1}^n \sigma_i$.

Transformations to/from for new observation data points. Dimensionality reduction transformations are directly following from SVD fold-in operation (2) described above.

Transformation of any new data point observation of an n-vector $\tilde{\mathbf{c}}_r$ *into* reduced dimensionality PCA space k-vector $\tilde{\mathbf{u}}$ will look like

$$\tilde{\mathbf{u}} = \boldsymbol{\Sigma}^{-1}\mathbf{V}^\top (\tilde{\mathbf{c}}_r - \boldsymbol{\xi}) \tag{7}$$

(this operation is essentially an SVD fold-in operation corrected for the mean subtraction). Note that, again, if input vectors tend to be quite sparse, then (7) could be decomposed as

$$\tilde{\mathbf{u}} = \boldsymbol{\Sigma}^{-1}\mathbf{V}^\top \tilde{\mathbf{c}}_r - \boldsymbol{\Sigma}^{-1}\mathbf{V}^\top \boldsymbol{\xi},$$

and online conversion can be sped up by precomputing term $\boldsymbol{\Sigma}^{-1}\mathbf{V}^\top\boldsymbol{\xi}$ which ends up to be a small dense k-vector, and big matrix $\mathbf{V}$ could be row-indexed for fast online matrix-vector multiplication.

Inverse transformation (from reduced PCA space into original space) looks like

$$\tilde{\mathbf{c}}_r = \boldsymbol{\xi} + \mathbf{V}\boldsymbol{\Sigma}\tilde{\mathbf{u}}. \tag{8}$$

MAHOUT-817 goals: why brute force approach is hard in context of big data computations. In context of massive computations, input $\mathbf{A}^{(pca)}$ is often rather sparse. Sparse matrices are packed and their subsequent operations within Mahout framework are optimized to account for degenerate nature of zero elements computation. Sometimes such reduction of need for flops and space may approach several orders of magnitude. However, mean subtraction step would turn such sparse inputs into a dense matrix $\left(\mathbf{A}^{(pca)} - \boldsymbol{\Xi}\right)$. Such intermediate input will take a lot of space and subsequent SVD will require a lot of flops. Fortunately, this can be addressed in context of SSVD method in a way that will be (almost) cost-equivalent to a regular SSVD computation on original sparse input $\mathbf{A}^{(pca)}$.

MAHOUT-817 addresses two goals:

- Provide column-wise mean computation step in the whole pipeline (or use outside mean vector if already available)
- Lift the dense matrix data concerns per above.

The sparser the original input is, the more efficiency gain is to be had by using SSVD PCA options compared to brute-force approach.

If original input is 100% dense, SSVD PCA options will have roughly the same cost as brute-force approach.

MAHOUT-817 introduces two additional pca options: `--pca` and `--pcaOffset` that request to treat incoming data as a PCA input. SVD rank opton `-k` will correspond to the reduced dimensionality of PCA space. MAHOUT-1067 introduces option `--uSigma`. In most cases where you might be looking to reduce dimensionality while retaining variance, you probably need combination of options `-pca true -U false -V false -us true`.[5] See §3 for details.

Finally, at the risk of complete disregard for reality we can do even rougher estimate as

$$\text{variance retained} = \frac{\sum_{i=1}^{k}\sigma_i}{\sum_{i=1}^{n}\sigma_i} \approx \frac{\sum_{i=1}^{k}\sigma_i}{0.5\,(n-k)\,\sigma_k + \sum_{i=1}^{k}\sigma_i}.$$

[5]Careful, option parsing in Mahout is tricky: just "-pca" is the same as "-pca false", i.e. no pca.

2 Input/output file formats and layout

Input $\mathbf{A}$, as well as outputs $\mathbf{U}\,(\mathbf{U}_\sigma)$, $\mathbf{V}\,(\mathbf{V}_\sigma)$ or $\mathbf{U\Sigma}$, are in Mahout's Distributed Row Matrix format, i.e. set of sequence files where value is of `VectorWritable` type. As far as keys are concerned, rows of $\mathbf{A}$ may be keyed (identified) by any `Writable` (for as long as it is instantiable thru a default constructor). That, among other thnigs, means that this method can be applied directly on the output of `seq2sparse` where keys are of `Text` type[6].

Definition of output $\mathbf{U}$ $(\mathbf{U}_\sigma)$ is identical to definition of the input matrix $\mathbf{A}$, and the keys of corresponding rows in $\mathbf{A}$ are copied to corresponding rows of output $\mathbf{U}$ $(\mathbf{U}_\sigma)$ or $\mathbf{U\Sigma}$. As of MAHOUT-1067, if row vectors of $\mathbf{A}$ are Mahout's named vectors, then corresponding output rows of the said outputs are also named vectors with the same names as row vectors of the input.

Definition of output $\mathbf{V}$ $(\mathbf{V}_\sigma)$ is always sequence file(s) of `(IntWritable, VectorWritable)` where key corresponds to a column index of the input $\mathbf{A}$.

Output of $\mathbf{\Sigma}$ is encoded by a single output file with a single vector value `(VectorWritable)` with main diagonal entries of $\mathbf{\Sigma}$ aka singular values $\begin{pmatrix}\sigma_1 & \cdots & \sigma_k\end{pmatrix}$.

[6] (TODO: re-verify)

3 CLI usage[7]

```
mahout ssvd <options>
```

Options.

`-k, --rank <int-value>` (required): the requested SVD rank (minimum number of singular values and dimensions in U, V matrices). The value of $k + p$ directly impacts running time and memory requirements. ***k+p=500 is probably more than reasonable***. Typically $k + p$ is taken within range 20...200.

`-p, --oversampling <int-value>` (optional, default 15): stochastic SVD oversampling. p doesn't seem to have to be very significant. If power iterations ($q > 0$) are used then p perhaps could be kept quite low, not to exceed 10% of k.

`-q, --powerIter <int-value>` (optional, default 0): number of power iterations to perform. This helps fighting data noise and improve precision significantly more than just increasing p. Each additional power iteration adds 2 more steps (map/reduce + map-only). Experimental data suggests using $q = 1$ is already producing quite good results which are hard to much improve upon.

`-t, --reduceTasks <int-value>` (required). The number of reducers to use. Recommended value for this option ~ 95% or ~190% of available reducer capacity to allow for opportunistic executions. However, this value would also depend on the task size: at some point increasing this value too much further may reach a point of diminishing returns or cause deficient blocking issue in reducers of `ABtJob` if chunks of works are too small. This is a super important parameter, hence it is now required.

`-r, --blockHeight <int-value>` (optional, default 10,000): the number of rows of source matrix for block computations during $\mathbf{Y} = \mathbf{QR}$ decomposition. Taller blocking causes more memory use but produces less blocks and therefore somewhat better running times. The most optimal mode from the running time point of view should be 1 block per 1 mapper. *This cannot be less than k+p.*

`-oh, --outerProdBlockHeight <int-value>` (optional, default 30,000): the block height during $\mathbf{B} = \mathbf{Q}^\top \mathbf{A}$ operation[8][9].

`-abth, --abtBlockHeight <int-value>` (optional, default 200,000): the block height during $\mathbf{Y}_i = \mathbf{A}\mathbf{B}_i^\top$ multiplication[8][9].

`-s, --minSplitSize <int-value>` (optional, default: use Hadoop's default): minimum split size to use in mappers reading $\mathbf{A}$ input. [10]

---

[7] As of Mahout 0.8 trunk.
12/13/2011 adjusted for MAHOUT-922 changes.
02/22/2012 MAHOUT-817.
10/07/2012 adjusted for MAHOUT-1067 changes.

[8] With extreme sparse matrices increasing this parameter may lead to better performance by reducing computational pressure on the shuffle and sort and grouping sparse records together.

[9] Watch for GC thrashing and swap. Values too high may cause GC thrashing and/or swapping, both of which are capable of bringing job performance down to a halt. Don't starve jobs for memory. Defaults are believed to work well for -Xmx800mb or above per child task.

[10] *As of this day, I haven't heard of a case where somebody would actually have to use this option and actually increase split size and how it has played out. So this option is experimental.*

Since in this version projection block formation happens in mappers, for a sufficiently wide input matrix the algorithm may not be able to read minimum $k + p$ rows and form a block of minimum height required, so in that case the job would bail out at

`-U, --computeU <true|false>` (optional, default true). Request computation of the U matrix

`-V, --computeV <true|false>` (optional, default true). Request computation of the V matrix

`-vhs, --vHalfSigma <true|false>` (optional, default: false): compute $\mathbf{V}_\sigma = \mathbf{V}\mathbf{\Sigma}^{0.5}$

`-uhs, --uHalfSigma <true|false>` (optional, default: false): compute $\mathbf{U}_\sigma = \mathbf{U}\mathbf{\Sigma}^{0.5}$

`-us, --uSigma<true|false>` (optional, default:false): compute product $\mathbf{U}\mathbf{\Sigma}$

`-pca<true|false>` (optional, default:false) run in pca mode: treat the input as $\mathbf{A}^{(pca)}$ and also compute column-wise mean $\boldsymbol{\xi}$ over the input and use it to compute PCA space (unless external column mean is already provided by `--pcaOffset`). (see §1.2)

`--pcaOffset <ξ-path>` (optional, default: none). Path(glob) of external column mean. The glob parameter must point to a sequence file containing single `VectorWritable` row as the $\boldsymbol{\xi}$ mean (see §1.2). This option can be used if column-wise mean is already efficiently obtained as byproduct from another pipeline, or if one wants to use custom centering offset for the data. This will save one MR pass over input since the mean will not have to be computed.

Standard Mahout options.

`--input <glob>` HDFS glob specification where the DistributedRowMatrix input to be found.

`--output <hdfs-dir>` non-existent hdfs directory where to output $\mathbf{U}, \mathbf{V}$ and $\mathbf{\Sigma}$ (singular values) files.

`--tempDir <temp-dir>` temporary dir where to store intermediate files (cleaned up upon normal completion). This is a standard Mahout optional parameter.

`-ow, --overwrite` overwrite output if exists.

---

the very first mapping step. If this happens, one of the recourses available is to force increase in the MapReduce split size using SequenceFileInputFormat.setMinSplitSize() property. Increasing this significantly over HDFS block size may result in network IO to mappers. Another caveat is that one sometimes does not want too many mappers because it may in fact increase time of the computation. Consequently, this option should probably be left alone unless one has significant amount of mappers (as in thousands of map tasks) at which point reducing amount of mappers may actually improve the throughput (just a guesstimate at this point).

4 Embedded use

4.1 SVD

It is possible to instantiate and use `SSVDSolver` class in embedded fashion in Hadoop-enabled applications. This class would have getter and setter methods for each option available via command line. See javadoc for details.

4.2 PCA

`SSVDSolver` is focused on SSVD solving primarily though, but if one looks to embed full PCA pipeline, an extra step is needed. One thing `SSVDSolver` doesn't know how to do is how to compute columns mean $\boldsymbol{\xi}$. Class `SSVDCli` (command line wrapper) uses `MatrixColumnMeansJob` for that purpose. Hence, full PCA pipeline is a two step pipeline: compute $\boldsymbol{\xi}$ + run pca-amended SSVD with it. See the code of command line wrapper `SSVDCli` for an example of how to run this 2 step solution.

5 FAQ

5.1 Small splits, small problems

"Can someone list all the constrains on the parameters (k,p &aBlockRows) that should be satisfied in order for the Q-job in ssvd to work fine? (...) I am getting the errors: "Givens thin QR: must be true: m>=n" or ""new m can't be less than n".

- Since we are in fact finding $k+p$ singlular values and singular vectors, general SVD requires rank $(\mathbf{A}) \geq k+p$. In more pragmatic terms this can be relaxed as follows: if $\mathbf{A}$ is input of $m \times n$ geometry, then input must satisfy
$$\min(m,n) \geq k+p.$$
This is also known as "problem too small" condition. All user issues fell into that category so far.
- In distributed version, blocking algorithms also require that *every split* of input $\mathbf{A}$ contains at least $k+p$ vectors.
  - In particular, this also means that in case of multiple files comprising $\mathbf{A}$, every input file of a distributed row matrix must contain at least $k+p$ row vectors. This is also the case of "problem too small" and requires consolidating several small files into a bigger input file so that split constraint holds.
  - It may be the case that *some* splits still have too few rows. However, in practice I never saw that happening, provided all above "problem too small" issues are not the case. However, assuming this is an issue, one need to adjust hadoop to use bigger splits (perhaps by trying bigger values for `-s` option than default split size). However, increasing splits may increase I/O and if it is a problem, perhaps using hdfs with a bigger split size is a better option to address this. (Again, remember that I still never encountered that case so that is quite unlikely).

- Similar to the reasoning in previous bullet, if $q > 0$ is used (power iterations), same problem theoretically *may* appear in reducers of ABtJob *if* any reducer task receives less than $k+p$ rows and unable to form vertical blocks with minimum rank required for the computation. However, since $k+p \sim 20..200$, it is also a *problem-too-small* case. In practice, I never saw this actually happening, if it happens, you probably do not need as many reducers as was claimed by -t option, it will not provide any tangible speed benefit on such short data streams.

5.2 Split idempotency of input A

Every time hadoop job attempts to create splits of input $\mathbf{A}$, it has to happen idempotently (which is basically a function of `SequenceFileInputFormat`). This is always the case with hadoop versions as of the time of this writing though.